5. How an IDOR Attack Happens?

An **IDOR (Insecure Direct Object Reference) attack** happens when an attacker takes advantage of insufficient access control on identifiers within an application to access or manipulate resources they shouldn't have permission for. Here's a step-by-step breakdown of how an IDOR attack typically unfolds:

1. Application Exposes Direct Object References

- Many applications use **direct identifiers** (e.g., IDs or file names) to reference resources, such as:

  ```
  https://example.com/profile?user_id=123
  ```

- These identifiers directly map to resources in the backend, such as database entries or files. If an application doesn't verify ownership, anyone can potentially access these resources by changing the identifier.

2. Attacker Identifies Vulnerable URLs or Parameters

- The attacker finds URLs or API endpoints where object references (like `user_id`, `file_id`, `order_id`) are used as parameters.
- These identifiers often follow a predictable sequence, making it easy for an attacker to guess or test different values.

3. Testing for Unauthorized Access

- The attacker modifies the identifier in the URL or parameter. For example, if they're logged in as `user_id=123`, they might change it to:

  ```
  https://example.com/profile?user_id=456
  ```

- By changing this parameter, the attacker is effectively trying to access another user's profile.

4. Gaining Unauthorized Access if No Checks are in Place

- If the application lacks proper access control verification, it will respond as if the request is legitimate and serve the requested resource, such as another user's profile, document, or transaction history.
- This happens because the application fails to check if the logged-in user is authorized to view or modify the requested resource.

5. Exploiting the Vulnerability

- After confirming the vulnerability, an attacker can exploit it to view, edit, or delete sensitive data. Common examples include:
  - Accessing another user's profile or financial information.
  - Downloading private documents by changing file identifiers.

- Altering other users' data by modifying IDs in forms or URLs.

Real-World Example: Accessing Another User's Order History

Imagine an e-commerce application that lets users view their orders using URLs like:

```
https://store.com/order?order_id=5001
```

An attacker could:

1. Log in to their own account and open one of their order pages.
2. Change the `order_id` in the URL to another value, like `5002`, which might belong to another user.
3. If the application doesn't check ownership, they could view or even alter another customer's order details.

Why IDOR Attacks Are Dangerous

- **Sensitive Data Exposure**: Attackers can gain unauthorized access to sensitive information.
- **Data Manipulation**: Attackers could modify, delete, or act on behalf of other users.
- **Harder to Detect**: Since IDOR doesn't necessarily generate suspicious activity (like a brute-force attack might), it can go undetected.

Mitigation Steps

To prevent IDOR vulnerabilities, applications should:

- **Implement Access Controls**: Always verify that a user has permission to access or modify a resource.
- **Use Indirect References**: Replace direct identifiers (like `user_id=123`) with indirect, non-guessable references (like UUIDs).
- **Perform Regular Security Audits**: Consistently test for access control issues to catch potential IDOR vulnerabilities early.

In short, an IDOR attack relies on insufficient checks on internal references, allowing attackers to easily gain unauthorized access simply by tweaking parameters.